Übungen zum Kapitel 5

Einführung in Pandas

Erstellt und überarbeitet: armin.baenziger@zhaw.ch, 16. Januar 2020

In [1]:
```
%autosave 0
```

```
Autosave disabled
```

Bevor wir beginnen, laden wir nötige Bibliotheken mit den üblichen Abkürzungen und erstellen zwei Listen und ein Dict:

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
list1 = [1, -2, 3, 0, 5, 2]
list_bool = [True, True, False, True, False, False]
dict1 = {'a': 100, 'b': 120, 'd': 95,
         'e': 105, 'f': 80, 'g': 105,}
```

(A.1) Erstellen Sie eine Series mit Name `ser1` mit den Werten aus Liste `list1`.

In [4]:
```
ser1 = pd.Series(list1)
ser1
```

Out[4]:
```
0    1
1   -2
2    3
3    0
4    5
5    2
dtype: int64
```

(A.2) Ändern Sie den Index von `ser1` in fortlaufende Kleinbuchstaben `a, b, ..., f` mit der Befehlszeile `ser1.index = list('abcdef')`

In [5]:
```
ser1.index = list('abcdef')
ser1
```

Out[5]:
```
a    1
b   -2
c    3
d    0
e    5
f    2
dtype: int64
```

(A.3) Lesen Sie den Wert zum Index `c` aus der Series `ser1` aus.

In [6]:
```
ser1['c']
```

Out[6]:
```
3
```

(A.4) Lesen Sie die ersten zwei Werte aus `ser1` aus.

In [7]:
```
ser1[:2]
# oder:
ser1[:'b']
```

Out[7]:
```
a    1
b   -2
dtype: int64
```

(A.5) Lesen Sie den letzten Wert aus `ser1` aus.

In [8]:
```
ser1[-1]
```

Out[8]:
```
2
```

(A.6) Lesen Sie die Werte zu den Indizes `b` und `e` (gleichzeitig) aus.

In [9]:
```
ser1[['b', 'e']]
```

Out[9]:
```
b   -2
e    5
dtype: int64
```

(A.7) Erstellen Sie eine Kopie der Series `ser1` und nennen Sie diese `ser1_c` . Führen Sie hierzu folgende Befehlszeile aus: `ser1_c = ser1.copy()`

In [10]:
```
ser1_c = ser1.copy()
```

(A.8) Ersetzen Sie den Wert zum Index `f` der Series `ser1_c` mit dem Wert `-2` .

In [11]:
```
ser1_c['f'] = -2
ser1_c
```

Out[11]:
```
a    1
b   -2
c    3
d    0
e    5
f   -2
dtype: int64
```

Folgende Anweisung erzeugt einen logischen Vektor mit `True` , falls der Wert negativ ist und ansonsten `False` .

In [12]:
```
ser1_c < 0
```

Out[12]:
```
a    False
b     True
c    False
d    False
e    False
f     True
dtype: bool
```

Mit folgender Anweisung lesen Sie alle negativen Werte aus `ser1_c` aus:

In [13]:
```
ser1_c[ser1_c < 0]
```

Out[13]:
```
b   -2
f   -2
dtype: int64
```

(A.9) Ersetzen Sie die negativen Werte in Series `ser1_c` mit `0`.

In [14]:
```
ser1_c[ser1_c < 0] = 0
ser1_c
```

Out[14]:
```
a    1
b    0
c    3
d    0
e    5
f    0
dtype: int64
```

(A.10) Prüfen Sie, ob in der Series `ser1` der Index `g` existiert. Verwenden Sie hierzu die Anweisung `'g' in ser1`.

In [15]:
```
'g' in ser1
```

Out[15]:
```
False
```

Betrachten Sie die Series `ser1` und die Liste `list_bool`.

In [16]:
```
ser1
```

Out[16]:
```
a    1
b   -2
c    3
d    0
e    5
f    2
dtype: int64
```

In [17]:
```
list_bool
```

Out[17]:
```
[True, True, False, True, False, False]
```

(A.11) Lesen Sie aus der Series `ser1` nur diejenigen Wert aus, für die die Liste `list_bool` an der entsprechenden Stelle `True` ist.

In [18]:
```
ser1[list_bool]
```

Out[18]:
```
a    1
b   -2
d    0
dtype: int64
```

(A.12) Erstellen Sie aus dem Dict `dict1` die Series `ser2`.

In [19]:
```
ser2 = pd.Series(dict1)
ser2
```

Out[19]:
```
a    100
b    120
d     95
e    105
f     80
g    105
dtype: int64
```

(A.13) Addieren Sie die zwei Series `ser1` und `ser2`. Speichern Sie das Resultat in der Series `resultat`. Was stellen Sie fest?

In [20]:
```
resultat = ser1 + ser2
resultat
# Die Werte werden für die Addition den Indizes zugeordnet.
# Es entstehen NaN, wenn ein Index in einer Series fehlt.
```

Out[20]:
```
a    101.0
b    118.0
c      NaN
d     95.0
e    110.0
f     82.0
g      NaN
dtype: float64
```

(A.14) Lesen Sie die Daten ohne `NaN` aus, indem Sie die Methode `notnull` entsprechend einsetzen. (Wir werden später dafür eine separate Methode kennenlernen, die die Auswahl nochmals vereinfacht.)

In [21]:
```
resultat[resultat.notnull()]
```

Out[21]:
```
a    101.0
b    118.0
d     95.0
e    110.0
f     82.0
dtype: float64
```

(A.15) Sortieren Sie `ser1` nach den Werten (nicht Index) aufsteigend (Methode `sort_values()`).

In [22]:
```
ser1.sort_values()
```

Out[22]:
```
b   -2
d    0
a    1
f    2
c    3
e    5
dtype: int64
```

(A.16) Sortieren Sie `ser1` nach den Werten in absteigender Reihenfolge. Tipp: Shif-Tab für Hilfe nach der ersten Klammer der Methode betätigen.

In [23]:
```
ser1.sort_values(ascending=False)
```

Out[23]:
```
e    5
c    3
f    2
a    1
d    0
b   -2
dtype: int64
```

Bevor wir weiterfahren erstellen wir eine Series und einen DataFrame:

In [24]:
```
np.random.seed(125)
n = 12

Zivilstand = np.random.choice(['l','v','g','vw'], size=n, replace=True)

dflohn = pd.DataFrame({'Personen_ID': (range(100,100+n)),
  'Lohn': np.random.lognormal(mean=8.6, sigma=0.4, size=n).astype(int),
  'Geschlecht': np.random.choice(['w','m'], size=n, replace=True),
  'Alter': np.random.randint(18, 66, n)})
```

(B.1) Lesen die die ersten fünf Fälle (Zeilen) von `dflohn` aus. Verwenden Sie hierzu die Methode `head()` .

In [25]:
```
dflohn.head()  # oder dflohn.head(5)
```

Out[25]:

	Personen_ID	Lohn	Geschlecht	Alter
0	100	1894	m	35
1	101	5888	w	56
2	102	4615	m	53
3	103	15911	w	35
4	104	3814	m	54

(B.2) Lesen Sie die letzten fünf Fälle (Zeilen) von `dflohn` aus. Verwenden Sie hierzu die Methode `tail()` .

In [26]:
```
dflohn.tail()
```

Out[26]:

	Personen_ID	Lohn	Geschlecht	Alter
7	107	8763	w	65
8	108	8681	w	40
9	109	5811	m	32
10	110	4462	m	46
11	111	3898	w	41

(B.3) Lesen Sie die Spalte `Lohn` aus dem DataFrame `dflohn` aus.

In [27]:
```
dflohn.Lohn    # oder: dflohn['Lohn']
```

Out[27]:
```
0      1894
1      5888
2      4615
3     15911
4      3814
5      4621
6      5657
7      8763
8      8681
9      5811
10     4462
11     3898
Name: Lohn, dtype: int32
```

(B.4) Setzten Sie als Index von `dflohn` die Spalte (Variable) `Personen_ID`. Führen Sie hierzu folgende Anweisung aus: `dflohn.set_index('Personen_ID', inplace=True)`. Was bewirkt das Argument `inplace=True`?

In [28]:
```
dflohn.set_index('Personen_ID', inplace=True)
dflohn.head()
# inplace=True führt dazu, dass der Index überschrieben wird,
# bzw. permanent ist.
```

Out[28]:

Personen_ID	Lohn	Geschlecht	Alter
100	1894	m	35
101	5888	w	56
102	4615	m	53
103	15911	w	35
104	3814	m	54

(B.5) Was bewirkt folgende Anweisung?

In [29]:
```
dflohn.rename(columns={'Lohn': 'Gehalt'}, inplace=True)
dflohn.head()    # Der Variablenname "Lohn" wurde in "Gehalt" geändert.
```

Out[29]:

Personen_ID	Gehalt	Geschlecht	Alter
100	1894	m	35
101	5888	w	56
102	4615	m	53
103	15911	w	35
104	3814	m	54

(B.6) Fügen Sie den (ebenfalls zuvor erstellten) Array `Zivilstand` dem DataFrame `dflohn` hinzu. Versuchen Sie hierzu folgende Anweisung: `dflohn['Zivilstand'] = Zivilstand`

In [30]:
```
dflohn['Zivilstand'] = Zivilstand
dflohn.head()
```

Out[30]:

Personen_ID	Gehalt	Geschlecht	Alter	Zivilstand
100	1894	m	35	v
101	5888	w	56	v
102	4615	m	53	g
103	15911	w	35	vw
104	3814	m	54	vw

(B.7) Lesen Sie die Daten der Person 104 aus dem Dataframe `dflohn` mit `loc` aus.

In [31]:
```
dflohn.loc[104]
```

Out[31]:
```
Gehalt        3814
Geschlecht       m
Alter           54
Zivilstand      vw
Name: 104, dtype: object
```

(B.8) Lesen Sie die Daten der Person 104 aus dem Dataframe `dflohn` mit `iloc` aus. Achtung, die *Position* der Person 104 ist (0, 1, 2, 3) **4**.

In [32]:
```
dflohn.iloc[4]    # Person 104 seht an Position 0, 1, 2, 3, *4*
```

Out[32]:
```
Gehalt        3814
Geschlecht       m
Alter           54
Zivilstand      vw
Name: 104, dtype: object
```

(B.9) Löschen Sie die Person 101 aus dem DataFrame `dflohn` durch `dflohn.drop(101, inplace=True)` .

In [33]:
```
dflohn.drop(101, inplace=True)
dflohn.head()
```

Out[33]:

Personen_ID	Gehalt	Geschlecht	Alter	Zivilstand
100	1894	m	35	v
102	4615	m	53	g
103	15911	w	35	vw
104	3814	m	54	vw
105	4621	m	29	vw

(B.10) Slicen Sie die Daten der Personen 103 bis und mit 106 aus dem DataFrame `dflohn` .

In [34]:
```
dflohn.loc[103:106]  # Achtung, hier inkl. 106, da 106 ein Label ist!
```

Out[34]:

Personen_ID	Gehalt	Geschlecht	Alter	Zivilstand
103	15911	w	35	vw
104	3814	m	54	vw
105	4621	m	29	vw
106	5657	m	59	g

(B.13) Slicen Sie die Variablen `Geschlecht` und `Zivilstand` der Personen 103 bis und mit 106 aus dem DataFrame `dflohn`. Versuchen Sie hieru die Anweisung `dflohn.loc[103:106, ['Geschlecht', 'Zivilstand']]`

In [35]:
```
dflohn.loc[103:106, ['Geschlecht', 'Zivilstand']]
```

Out[35]:

Personen_ID	Geschlecht	Zivilstand
103	w	vw
104	m	vw
105	m	vw
106	m	g

(B.14) Lesen Sie die Daten der *weiblichen* Personen aus dem DataFrame `dflohn` aus.

In [36]:
```
dflohn[dflohn.Geschlecht=='w']
```

Out[36]:

Personen_ID	Gehalt	Geschlecht	Alter	Zivilstand
103	15911	w	35	vw
107	8763	w	65	v
108	8681	w	40	g
111	3898	w	41	vw

(B.9) Ersetzen Sie das *Gehalt* von *Person 104* mit dem Fehlwert `NaN` (durch `np.nan` oder `None`).

In [37]:
```
dflohn.loc[104, 'Gehalt'] = None
dflohn.head()
```

Out[37]:

Personen_ID	Gehalt	Geschlecht	Alter	Zivilstand
100	1894.0	m	35	v
102	4615.0	m	53	g
103	15911.0	w	35	vw
104	NaN	m	54	vw
105	4621.0	m	29	vw

(C.1) Bestimmen Sie das maximale Gehalt im Dataframe `dflohn`.

In [38]:
```
dflohn.Gehalt.max()
```

Out[38]:
```
15911.0
```

(C.2) Wählen Sie die Zeile (Person) aus, die das maximale Gehalt bezieht.

In [39]:
```
dflohn[dflohn.Gehalt==dflohn.Gehalt.max()]
```

Out[39]:

Personen_ID	Gehalt	Geschlecht	Alter	Zivilstand
103	15911.0	w	35	vw

(C.3) Bestimmen Sie je das arithmetische Mittel der Variablen `Alter` und `Gehalt` in `dflohn`.

In [40]:
```
dflohn.mean()
```

Out[40]:
```
Gehalt    6431.300000
Alter       44.454545
dtype: float64
```

(C.4) Bestimmen Sie je den Median der Variablen `Alter` und `Gehalt` in `dflohn`.

In [41]:
```
dflohn.median()
```

Out[41]:
```
Gehalt    5139.0
Alter       41.0
dtype: float64
```

(C.5) Führen Sie den Befehl `dflohn.describe()` aus. In welcher Zeile befindet sich der Median?

In [42]:
```
dflohn.describe()
# Median = 50%-Quantil
```

Out[42]:

	Gehalt	Alter
count	10.000000	11.000000
mean	6431.300000	44.454545
std	3927.198239	11.852119
min	1894.000000	29.000000
25%	4500.250000	35.000000
50%	5139.000000	41.000000
75%	7963.500000	53.500000
max	15911.000000	65.000000

(C.6) Erstellen Sie eine Häufigkeitsverteilung der Variable `Zivilstand`. Tipp: Im Pandas-Cheat-Sheet unter "Summarize Data" nachschauen, falls Sie den Namen der Methode vergessen haben.

In [43]:
```
dflohn.Zivilstand.value_counts()
```

Out[43]:
```
vw    5
g     3
v     2
l     1
Name: Zivilstand, dtype: int64
```

Für den letzten Teil laden wir einige reale Aktienpreisdaten:

In [44]:
```
# Beispieldaten:
Preise = pd.read_pickle('../examples/yahoo_price.pkl')
Preise.head()
```

Out[44]:

Date	AAPL	GOOG	IBM	MSFT
2010-01-04	27.990226	313.062468	113.304536	25.884104
2010-01-05	28.038618	311.683844	111.935822	25.892466
2010-01-06	27.592626	303.826685	111.208683	25.733566
2010-01-07	27.541619	296.753749	110.823732	25.465944
2010-01-08	27.724725	300.709808	111.935822	25.641571

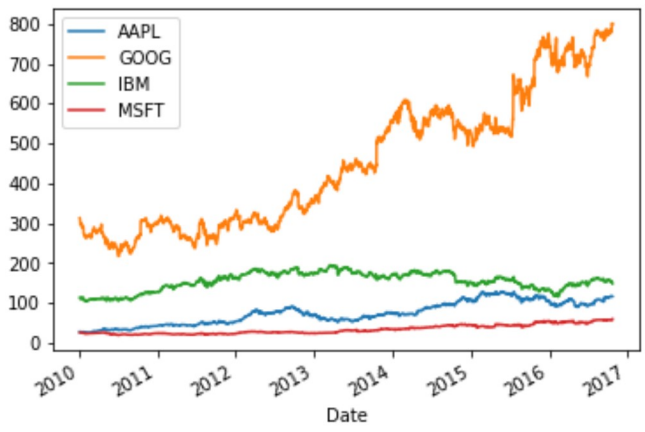
Damit Diagramme automatisch in den Zellen erscheinen, führen wir folgenden Magic Command aus:

In [45]:
```
%matplotlib inline
```

(D.1) Plotten Sie die Preisdaten mit `Preise.plot()`.

In [46]:
```
Preise.plot()
```

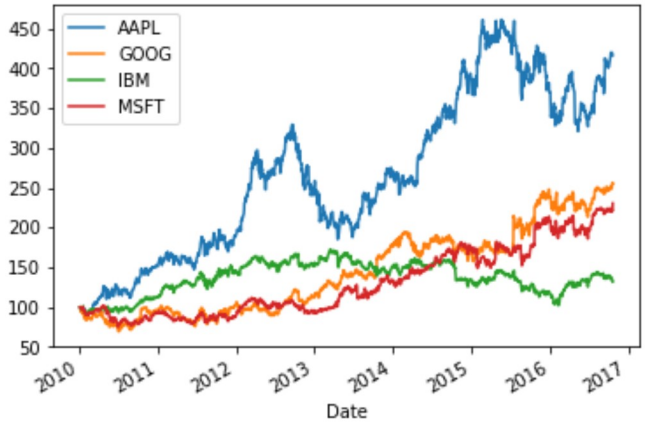
Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c01c3275c0>
```



(D.2) Für den besseren Vergleich der Performance sollten wir die Preisdaten umbasieren. Erstellen Sie das DataFrame `Preise100`, bei dem alle Aktienkurse mit 100 starten. Versuchen Sie hierzu die Anweisung `Preise100 = Preise/Preise.iloc[0]*100`. Plotten Sie die Kurese danach nochmals.

In [47]:
```
Preise100 = Preise/Preise.iloc[0]*100
Preise100.plot()
# Nun sieht man, dass im Beobachtungszeitraum die Performance
# von Apple am besten war.
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c01cb68358>
```



Die nun zu definierende Funktion `HPR()` berechnet die Gesamtrendite (Holding Period Return) in Prozent über den Beobachtungszeitraum für jede Aktie eines Preis-DataFrames.

In [48]:
```
def HPR(P):
    return (P.iloc[-1]/P.iloc[0] - 1)*100
```

(D.3) Verwenden Sie die Funktion `HPR()`, um die Gesamtrenditen aller Aktien im DataFrame `Preise` zu berechnen.

In [49]:
```
HPR(Preise)
```

Out[49]:
```
AAPL    316.573978
GOOG    155.338815
IBM      32.060031
MSFT    130.488952
dtype: float64
```

In [50]:
```
# Oder so:
Preise.apply(HPR)
```

Out[50]:
```
AAPL    316.573978
GOOG    155.338815
IBM      32.060031
MSFT    130.488952
dtype: float64
```

(D.4) Erstellen Sie den DataFrame `Renditen`, welcher die Tagesrenditen, berechnet aus den Preisdaten im DataFrame `Preise`, enthält. Verwenden Sie hierzu die nützliche Methode `pct_change()`.

In [51]:
```
Renditen = Preise.pct_change()
Renditen.head()
```

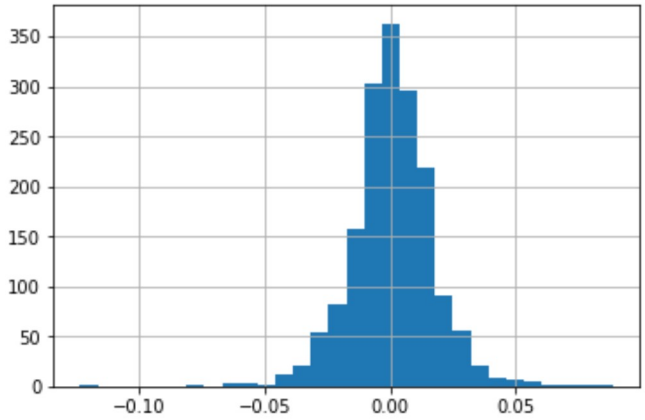
Out[51]:

Date	AAPL	GOOG	IBM	MSFT
2010-01-04	NaN	NaN	NaN	NaN
2010-01-05	0.001729	-0.004404	-0.012080	0.000323
2010-01-06	-0.015906	-0.025209	-0.006496	-0.006137
2010-01-07	-0.001849	-0.023280	-0.003462	-0.010400
2010-01-08	0.006648	0.013331	0.010035	0.006897

(D.5) Erstellen Sie ein Histogramm der Tagesrenditen der Aktie von Apple mit 30 Klassen (bins). Versuchen Sie hierzu die Anweisung `Renditen.AAPL.hist(bins=30)` .

In [52]:
```
Renditen.AAPL.hist(bins=30)
```

Out[52]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c01cbd5ba8>
```



(D.6) Mit der Methode `quantile` , welche auf Series und DataFrames angewendet werden kann, können Quantile von Verteilungen bestimmt werden. Finden Sie von allen vier Titeln das 1%- und 5%-Quantil der Renditen (z. B. für nicht-parametrische Value-at-Risk-Berechnungen).

In [53]:
```
Renditen.quantile([0.01, 0.05])
```

Out[53]:

	AAPL	GOOG	IBM	MSFT
0.01	-0.042284	-0.040530	-0.036046	-0.038141
0.05	-0.025154	-0.021922	-0.017780	-0.021171

Hinweis zur Auswertung: Das 5%-Quantil von Apple beträgt etwa -2.5%. Dies bedeutet, dass etwa 5% der Tagerenditen unter -2.5% und 95% der Tagesrenditen darüber lagen.

(D.7) Berechnen Sie die Korrelation zwischen den Renditen von Apple (`AAPL`) und Google (`GOOG`) auf drei Nachkommastellen genau.

In [54]:
```
Renditen.AAPL.corr(Renditen.GOOG).round(3)
```

Out[54]:
```
0.408
```

Ende der Übung